# Classes and Interfaces

## Summary

- Object-oriented programming is one of the many programming paradigms (styles of programming) in which objects are the building blocks of applications.
- An object is a unit that contains some data represented by properties and operations represented by methods.
- A class is a blueprint for creating objects. The terms class and object are often used interchangeably.
- We use access modifiers (public, private, protected) to control access to properties and methods of a class.
- A constructor is a special method (function) within a class that is called when instances of that class are created. We use constructors to initialize properties of an object.
- Static members are accessed using the class name. We use them where we need a single instance of a class member (property or method) in memory.
- Inheritance allows a class to inherit and reuse members of another class. The providing class is called the *parent*, *super* or *base* class while the other class is called the *child*, *sub* or *derived* class.
- An abstract class is a class with partial implementation. Abstract classes cannot be instantiated and have to be inherited.
- We use interfaces to define the shape of objects.

 

# Cheat Sheet

### Classes and constructors

```
class Account {
  id: number;

  constructor(id: number) {
    this.id = id;
  }
}

let account = new Account(1);
```

### Accessing properties and methods

```
account.id = 1;
account.deposit(10);
```

### Read-only and optional properties

```
class Account {
  readonly id: number;
  nickname?: string;
}
```

### Access modifiers

```
class Account {
  private _balance: number;

  // Protected members are inherited.
  // Private members are not.
  protected _taxRate: number;
}
```

### Parameter properties

```
class Account {
  // With parameter properties we can
  // create and initialize properties in one place.
  constructor(public id: number, private _balance: number) {
  }
}
```

### Getters and setters

```
class Account {
  private _balance = 0;

  get balance(): number {
    return this._balance;
  }

  set balance(value: number) {
    if (value < 0)
      throw new Error();
    this._balance = value;
  }
```

### Index signatures

```
class SeatAssignment {
  // With index signature properties we can add
  // properties to an object dynamically
  // without losing type safety.
  [seatNumber: string]: string;
}

let seats = new SeatAssignment();
seats.A1 = 'Mosh';
seats.A2 = 'John';
```

 

### Static members

```
class Ride {
  static activeRides = 0;
}

Ride.activeRides++;
```

### Inheritance

```
class Student extends Person {
}
```

### Method overriding

```
class Student extends Person {
  override speak() {
    console.log('Student speaking');
  }
}
```

### Abstract classes and methods

```
abstract class Shape {
  // Abstract methods don't have a body
  abstract render();
}

class Circle extends Shape {
  override render() {
    console.log('Rendering a circle');
  }
}
```

 

### Interfaces

```
interface Calendar {
  name: string;
  addEvent(): void;
}

class GoogleCalendar implements Calendar {
}
```

 

# Compiler Options

| Option | Description |
| --- | --- |
| `noImplicitOverride` | When enabled, then compiler will warn us if we try to override a method without using the override keyword. |

 